ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

УДК 336.76

СОГЛАСОВАНО
Руководитель проекта,
преподаватель
департамент больших данных и информационного поиска

_____ П. П. Лукьянченко
«___» _____ 2021 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия»,
профессор департамента программной инженерии, канд. техн. наук
_____ В. В. Шилов
«___» _____ 2021 г.

Отчет

к курсовому проекту

на тему «Предсказание ликвидности рынка»

по направлению подготовки бакалавров 09.03.04 «Программная инженерия»

Выполнила
студентка группы 194
образовательной программы
09.03.04 «Программная инженерия»

Ю. В. Ловчикова

_____
Подпись, Дата

Москва 2021

Содержание

Реферат

Отчет 36 стр., 7 рис., 1 прил., 5 частей отчёта, 7 источников.

Ключевые слова: *ликвидность, нейронные сети, финансовые рынки, предсказание временных рядов, LSTM модель нейронной сети.*

В отчете представлены результаты курсовой работы на тему «Предсказание ликвидности рынка», выполненной на основе приказа Национального исследовательского университета "Высшая школа экономики".

Объект исследования – финансовый рынок (криптовалют).

Предмет исследования – нейросетевой прогноз временных рядов.

Цель исследования – создание программы по предсказанию показателей рынка, определяющих ликвидность.

Методы исследования:

1. Прохождение курса Deep Learning School по машнному обучению и нейронным сетям
2. Изучение публикаций и статей
3. Сравнительный анализ
4. Машинное обучение с использованием нейросетевой модели LSTM

Результаты работы:

1. Изучено понятие ликвидности и выявлена неявная функция от нескольких параметров, задающая ликвидность
2. Изучены принципы машинного обучения и нейрнонных сетей
3. Проведён сравнительный анализ моделей нейронных сетей для использования в предсказании временных рядов
4. Создан класс для дальнейшего использования с целью предсказания параметров рыночной ликвидности

Определения

1. Ликвидность – рыночная харакетристика, более подробное описание см. в разделе «Понятие ликвидности»
2. Спред - разность между лучшими ценами заявок на продажу (аск) и на покупку (бид) в один и тот же момент времени на какой-либо актив (акцию, товар, валюту, фьючерс, опцион)
3. Рынок - совокупность процессов и процедур, обеспечивающих обмен между покупателями и продавцами отдельными товарами и услугами.

Введение

Обоснование необходимости проведения данной научно-исследовательской работы и актуальность темы

Эффект ликвидности в торговле хорошо заметен, причём для всех участников процессов на финансовом рынке.

Рациональный инвестор всегда предпочел бы торговать в актив с большой ликвилностью, и есть повсеместные свидетельства корреляции между более высокими объемами торгов и более высокой рыночной ликвидностью. С другой стороны, состояние низкой ликвидности обычно приводит к более низким ценам на активы, что менее выгодно для инвесторов, которым в этом случае потребуется более высокая доходность, чтобы компенсировать хранение неликвидных активов. Чаще всего ликвидность рынка постоянно меняется, и поэтому инвесторы могут потребовать дополнительных компенсаций за подверженность такому риску ликвидности. По этой причине было бы разумно предположить, что все инвесторы принимают во внимание ликвидность в торговле и соответственно строят торговые стратегии. Помимо вышеупомянутых торговых проблем, ликвидность помогает нам понять цену активов. Например, в эмпирических исследованиях ликвидность хорошо объясняет перекрестную доходность активов, имеющих разную ликвидность, а также взаимосвязь между доходностью ликвидности и капитала в измерении временного ряда.

Ликвидность представляют интерес не только для инвесторов, но и для центральных банков, поскольку они имеют последствия как для денежно-кредитной, так и для финансовой стабильности. Всё чаще трансграничные связи означают, что дисбаланс ликвидности в одной стране может привести к вторичным эффектам в экономике другой страны. Это наиболее очевидно в случае более широкого определения ликвидности: инвесторы, сталкивающиеся с нехваткой внутренних активов (избыточная ликвидность также может рассматриваться как нехватка активов) могут попытаться получить доступ к иностранным активам; обычно это касается валютных рынков и трансграничных операций. В той мере, в какой балансы центрального банка и коммерческих банков взаимозаменяемы на оптовых денежных рынках, неравновесие в крупных деньгах может также распространяться за границу, о чем свидетельствует поведение рынков в Соединенных Штатах, Великобритании и зоне евро в августе – сентябре 2007 г.

Таким образом, ликвидность – ценный и важный показатель рынка, и важно уметь его прогнозировать.

Новизна темы

В ходе работы была разработана нейронная сеть, прогнозирующая временной ряд нужных биржевых праметров, но помимо этого был проведён анализ эффективности предсказаний и их точности. Иными словами, данная научная работа заключается в анализе возможного практического применения LSTM модели нейросети.

Понятие ликвидности

Термин «ликвидность» используется в трех разных условиях. Во-первых, ликвидность может обозначать ликвидность фирмы, также называемую **платежеспособностью**. Во-вторых, ликвидность - это характеристика актива, также называемая «ликвидностью актива» или «рыночной ликвидностью», в зависимости от того, находится ли в фокусе баланс или рынок. С точки зрения инвестора он описывает **конкурентоспособность или «простоту торговли активом»**. В-третьих, ликвидность также используется **с денежно-кредитной точки зрения** и касается ликвидности всей экономики. Мы будем рассматривать рыночную ликвидность, потому что она несёт больший интерес для массового точечного применения инвесторами и брокерами.

Наиболее часто упоминаемые параметры ликвидности - это герметичность, оперативность, глубина, широта и устойчивость:

(i) герметичность (tightness): Герметичность относится к низким операционным издержкам, таким как разница между ценами покупки и продажи, например, спреды между ценами покупки и продажи на рынках, основанных на котировках, а также неявные затраты.

(ii) оперативность (immediacy): Оперативность представляет собой скорость, с которой приказы могут быть выполнены и, в этом контексте, также урегулированы, и, таким образом, отражает, среди прочего, эффективность торговых клиринговых и расчетных систем.

(iii) глубина (depth): Под глубиной понимается наличие большого количества реальных или легко обнаруживаемых заказов потенциальных покупателей и продавцов как выше, так и ниже цены, по которой сейчас торгуется ценная бумага.

(iv) широта (breadth): Широта означает, что заказы одновременно многочисленны и имеют большой объем с минимальным влиянием на цены.

(v) устойчивость (resiliency): Устойчивость - это характеристика рынков, на которых новые заказы поступают быстро, чтобы исправить дисбаланс заказов, что, как правило, уводит цены от того, что гарантируется фундаментальными факторами.

Эти условия отражают различные аспекты того, *насколько быстро и без значительных затрат* актив может быть преобразован в законное платежное средство.

Для того, чтобы понять, как на практике оценивать изменение ликвидности, нужно представить перечисленные параметры в исчислимых величинах. Трудностью в этом вопросе становится неполная информация во многих свободных датасетах, и поэтому, опираясь на имеющиеся возможности, мы выявили следующие параметры, по которым оценивали рыночную ликвидность:

1. Бид-аск спред – показатель герметичности
2. Объём продажи – показатель широты
3. Цена сделки – показатель глубины
4. Время транзакции – показатель оперативности

Показатели 2, 3 связаны некоторой прямой зависимостью с функцией ликвилности, а показатели 1, 4 – обратной. Но важно отметить, что не существует явной функции ликвидности, и можно лишь оценить масштабы её изменений в течение времени, поскольку многие посторонние экзогенные факторы могут сильно влиять на поведение ликвидности. Это влечёт за собой предположение о том, что ликвидность трудно оценивается программными средствами, и не в каждый период времени может давать близкий к реальному результат.

Нейросетевая модель и логика работы программы

Используемая библиотека и язык

Для разработки кода использовался фреймворк PyTorch машинного обучения для языка Python с открытым исходным кодом, созданный на базе библиотеки Torch. Для разработки нейронных сетей используется также библиотека TensorFlow языка Python, но выбранная библиотека содержит большее количество средств для математической оценки работы модели.

Модель нейросети

В качестве модели нейросети была выбрана LSTM.

LSTM – один из наиболее продвинутых представителей рекуррентных нейронных сетей, которые отличаются от других тем, что используют знания о предыдущих событиях, чтобы изучить последующие. Такой принцип работы наиболее удачен для работы с временными рядами. Преимущество LSTM перед другими моделями заключается в том, что она эффективна даже при большом разрыве между блоками информации, а обычные рекуррентные нейросети влекут в таком случае сильное увеличение ошибки. Другими словами, LSTM способны обучаться долгосрочным зависиомстям.

Для сравнения приведём схематичное представление принципа работы рекуррентных нейронных сетей и LSTM сетей:

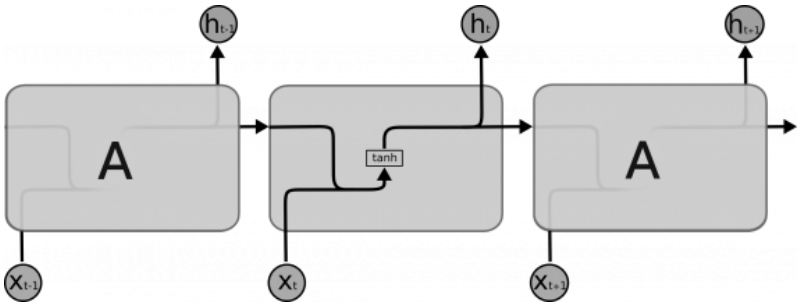

Рис.1, Рекуррентная нейросеть

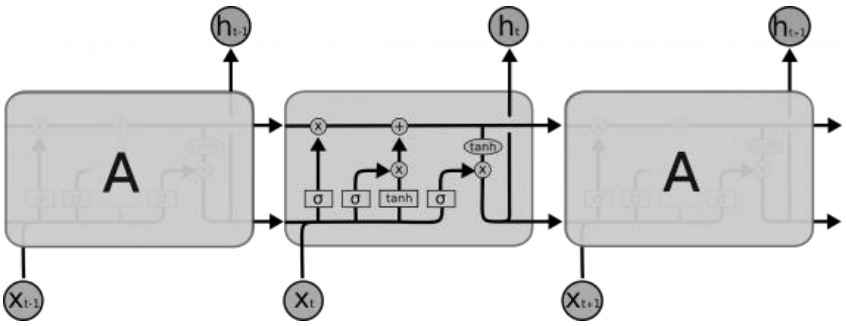

Рис. 2, LSTM нейросеть

Как можно видеть, LSTM содержит больше блоков для удержания и обработки данных.

Логика работы программы

1. Подготовка датасета для обучения

Для загрузки датасета был выбран ресурс https://www.cryptodatadownload.com/, по той причине, что он содержит большой объём данных и показатели сделок с разницей в несколько секунд.

Поскольку для эффективной работы нейросети необходима обработка данных, а многие доступные биржевые датасеты не содержат полной информации для анализа поведения ликвидности, мы прибегнули к некоторым методам извлечения информации, которые будут описаны далее.

После сортировки данных по времени осталось решить три проблемы: извлечь данные о спреде, извлечь данные о времени транзакции, сформировать датасет с равными промежутками времени между транзакциями. Время транзакции полагали как разность между двумя соседними сделками, бид-аск спред полагали как разница между соседними сделками с пометками “sell” и “buy”, а для выравнивания промежутков выбрали интервал в 1 минуту и усреднили значения на каждом промежутке. Важно понимать, что эти значения не являются натуральными, но их можно иметь в виду как статистически подходящие.

Далее был проведён анализ корреляции признаков между собой и актуальности предстоящего обучения для имеющихся рядов:

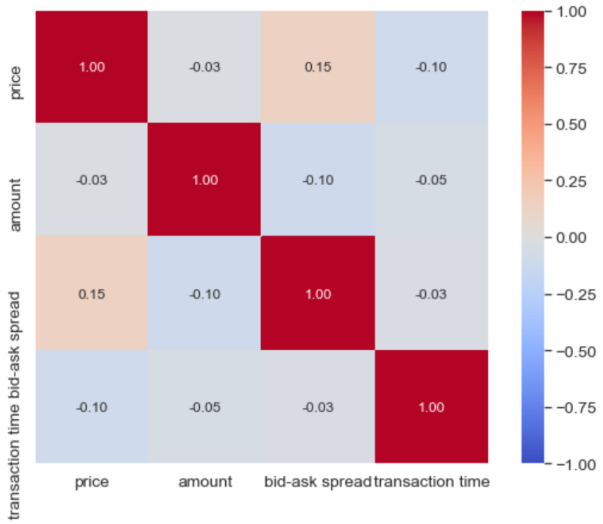

Рис. 3, Коэффициенты корреляции пар признаков

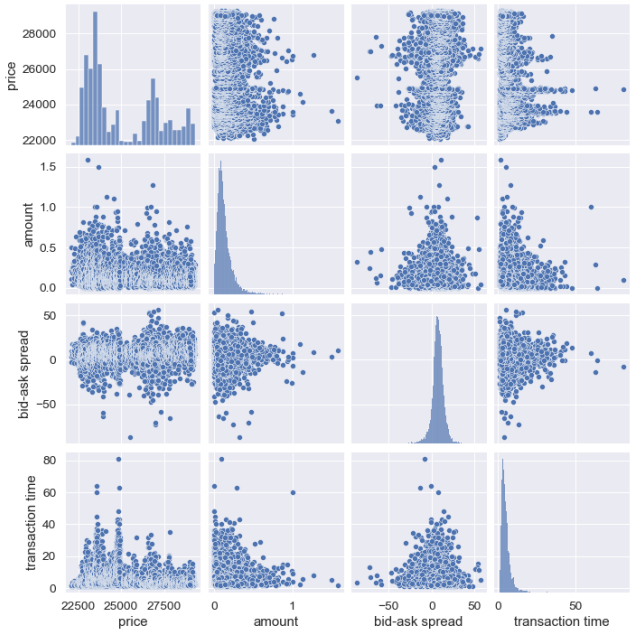

Рис. 4, Графики попарных зависимостей признаков

Некоторая корреляция признаков прослеживается, но она нормальна, поэтому нейросеть должна показать независимые результаты.

Для просмотра зависимости каждого временного ряда использовались первые 100 значений датасета:

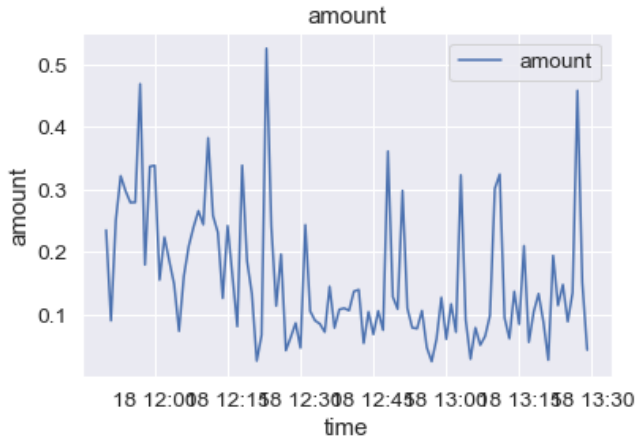

Рис. 4, Временной ряд параметра объёма сделок

Рис. 4, Временной ряд параметра цены сделок

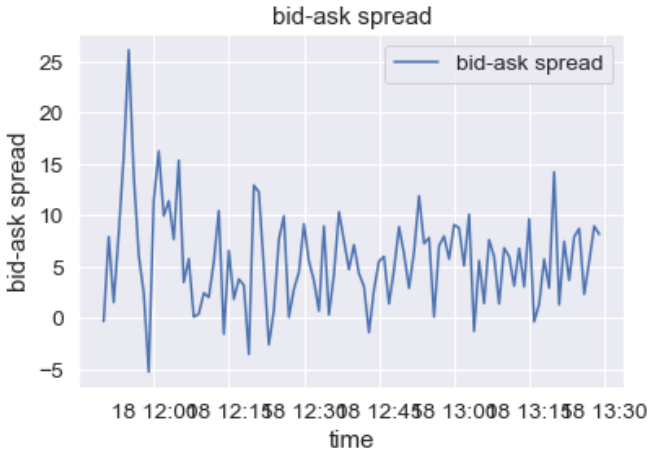

Рис. 4, Временной ряд параметра спреда сделок

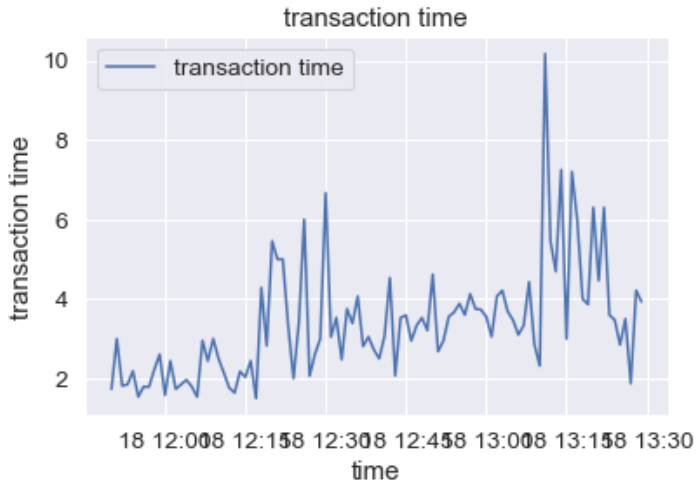

Рис. 4, Временной ряд параметра времени транзакций

Судя по графикам, наиболее четкая зависимость прослеживается у параметров объёма и спреда, поэтому, скорее всего, именно они дадут более чёткие прогнозы.

Далее каждый временной ряд был поделён на две части: первая (66%) – для обучения модели, вторая (33%) – для тестирования корректности. Каждый массив был поделён на батчи размеров 60 элементов (из соображений количества минут в часе), и для каждого батча были выделены labels элементы, с которыми сравнивались результаты обучения или тестирования. Далее все данные были нормированы в пределах [-1;1] для успешной работы нейросети.

2. Подготовка данных для прогнозирования

Для прогнозирования был выбран датасет меньшего размера, с которым была проделана та же подготовительная работа, что и с данными для обучения и тестирования, кроме выделения labels-элементов.

3. Обучение, тестирование и прогнозирование

Результаты работы нейронной сети

1. Оценка ошибки на основе тестирования

   (i) Цена

   MAE: 0.37914, RMSE: 0.43922, R2: 0.33697

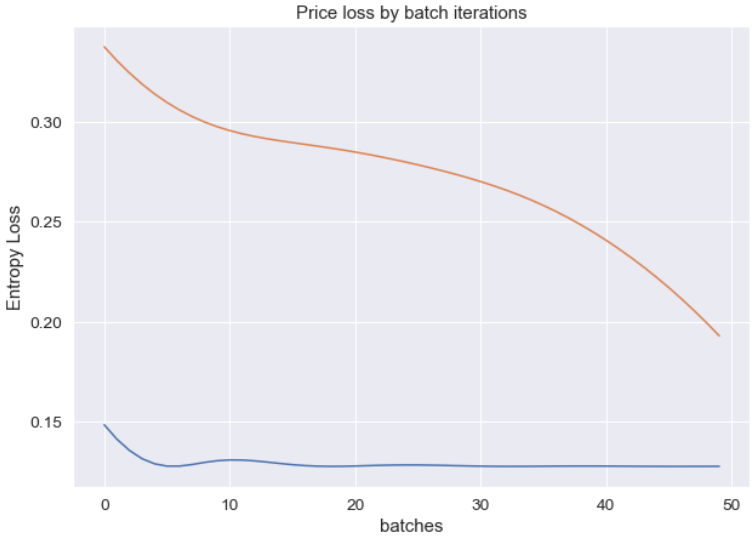

Рис. 5, График функции потерь для предсказанных и действительных значений параметра цены

   (ii) Объём

   MAE: 0.23877, RMSE: 0.25903, R2: -2.30421

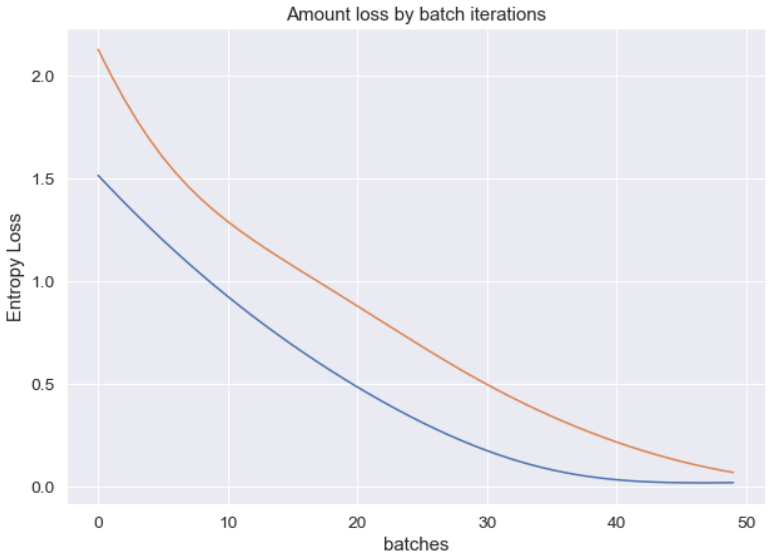

Рис. 6, График функции потерь для предсказанных и действительных значений параметра объёма

(iii) Бид-аск спред

MAE: 0.08262, RMSE: 0.11785, R2: 0.04715

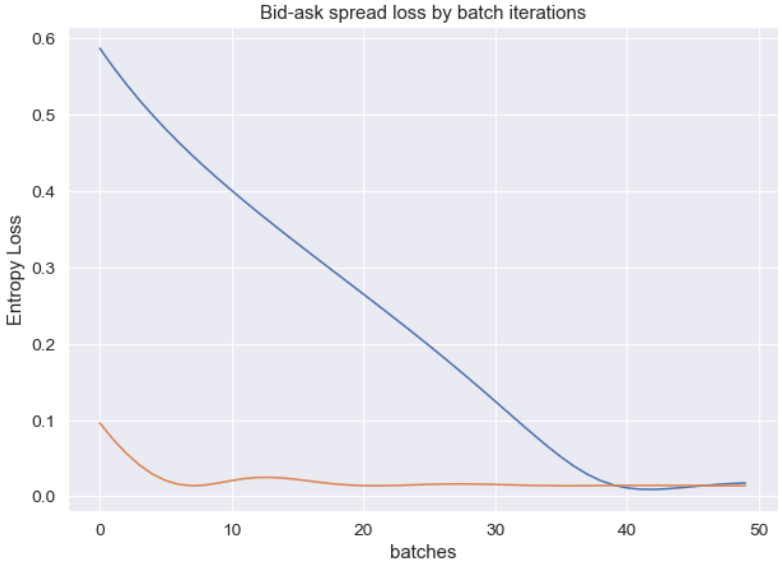

Рис. 6, График функции потерь для предсказанных и действительных значений параметра спреда

(iv) Время транзакции

MAE: 0.09968, RMSE: 0.13486, R2: 0.02476

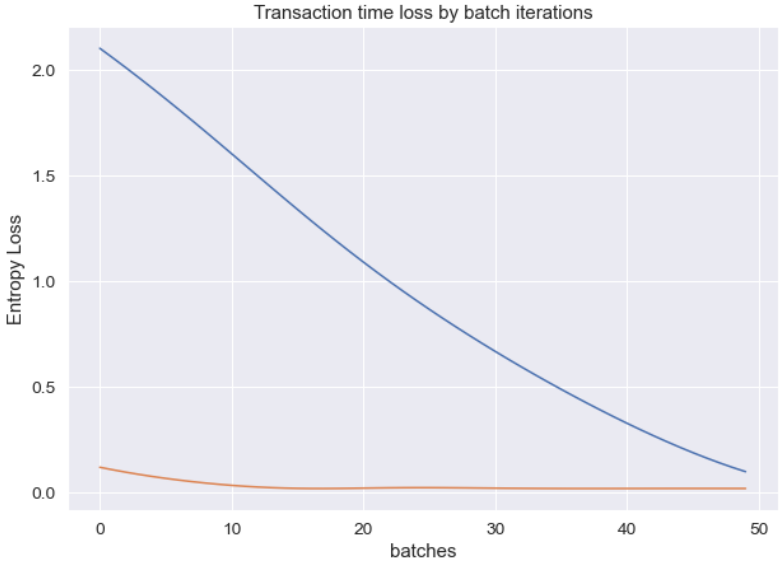

Рис. 7, График функции потерь для предсказанных и действительных значений параметра времени транзакции

По полученным графикам и рассчитанным значениям ошибок можно заключить, что наиболее точные предсказания будут получены для параметров бид-аск спреда и объёма продаж, но для других параметров нельзя утверждать точность предсказанных значений.

Заметим, что в предыдущем разделе данного Отчёта «Нейросетевая модель и логика работы программы» в пункте 1 «Подготовка датасета для обучения» подраздела «Логика работы программы» был проведён предварительный анализ данных, где прогноз на большую точность предсказанных данных относился к тем же параметрам, какие и имеют меньшее отклонение от реальных значений в ходе тестирования, а именно спред и объём. Таким образом, по предварительному анализу можно судить о валидности датасета для обучения.

2. Предсказанные значения

(i) Цена

Значение в последней точке: 10.87499505

Предсказанное значение в следующей точке: 35.31283474

(ii) Объём

Значение в последней точке: 4.02857876

Предсказанное значение в следующей точке: 18.04008192

(iii) Бид-аск спред

Значение в последней точке: 61.00410154

Предсказанное значение в следующей точке: 57.99638233

(iv) Время транзакции

Значение в последней точке: 1.90909183

Предсказанное значение в следующей точке: 10.01324958

Первые три параметра говорят об увеличении ликвидности, а последний – наоборот. Но по полученным результатам можно предсказывать динамику ликвидности, и, опираясь на это, брокеры могут совершать решения относительно определённых активов и акций.

Заключение

В результате работы было выявлено следующее:

1. С теоретической точки зрения, для анализа временных рядов рынка более подходящей является LSTM модель.
2. По предварительному анализу можно судить о валидности датасета для обучения.
3. Точно определить характер поведения ликвидности не получится, но можно предсказать её динамику.

Таким образом, можно утверждать о пользе использования нейронных сетей для прогнозирования поведения ликвидности.

Направления для дальнейшей работы:

1. Провести сравнительный анализ различных моделей нейронных сетей на одном и том же наборе данных.
2. Найти новые численные метрики для прочих параметров, влияющих на ликвидность.

Список источников

1. Market Making with Machine Learning Methods / Kapil Kanagal, Yu Wu, Kevin Chen. June 10, 2017. – 13 с.
2. Huang, Yuping (2015) Liquidity in equity markets. PhD thesis. / Huang, Yupin. - Adam Smith Business School University of Glasgow, October, 2015. – 224 с.
3. Market liquidity risk: an overview / Stange, Sebastian; Kaserer, Christoph. Center for Entrepreneurial and Financial Studies, April 7, 2009. – 35 с.
4. Measuring Liquidity in Financial Markets. / Tonny Lybek, Abdourahmane Sarr. International Monetary Fund, December, 2002. – 63 с.
5. Liquidity metrics improve understanding of portfolio risk / Philip Higson, Marius Müller. Carlyon AG, July 2019. – 28 с.
6. Liquidity forecasting / Simon T Gray. Bank of England, 2008. – 33 с.
7. Курс «Deep Learning» на платформе Stepik [Электронный ресурс] - https://stepik.org/course/91156/syllabus

Приложение 1

Код программы

1. Загружаем данные для обучения

```
#!/usr/bin/env python
# coding: utf-8

# In[319]:


import pandas as pd
import numpy as np
from matplotlib import pyplot as plt


# In[449]:


data = pd.read_csv('BTCUSD_Bitstamp_Q4_2020_prints.csv')
data
```

2. Готовим данные для обучения и тестирования
   2.1. Чистим имеющиеся данные и добавляем новый столбец «время транзакции»

```
data=data.drop('trans_id',1)
data=data.drop('dollar_amount',1)
data=data.drop('unix',1)


# In[451]:


data['symbol'].value_counts()


# Поскольку валюта только одна, то и этот столбец можем не брать во внимание

# In[452]:


data=data.drop('symbol',1)


# In[453]:


data.dtypes


# In[454]:
```

```
# Откорректируем формат данных столбца с датами
from datetime import datetime
data['date']=data['date'].apply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S"))


# In[455]:


data.dtypes


# In[456]:


# Сортируем данные от более старых к более новым
data=data.sort_values(by='date')


# In[457]:


data['bid-ask spread']=np.nan


# In[458]:


first=data['date'].to_numpy()
second=np.roll(data['date'],1)


# In[459]:


data['transaction time']=(first-second)


# In[460]:


data=data.iloc[1:]
data['transaction time']=data['transaction time'].apply(lambda x: x.total_seconds())


# In[462]:


data.index = range(0,len(data))


# In[463]:


data=data.iloc[-300000:]
data.index = range(0,len(data))
```

```
# In[464]:


data.dtypes
```

2.2. Исследуем, как получить бид-аск спред, и добавляем соответствующий столбец

```
# In[465]:


get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()


# In[466]:


# Для того, чтобы не засорять вывод предупреждениями
import warnings
warnings.filterwarnings('ignore')


# In[467]:


data1=data.iloc[:100]
price_ = data1[['type','price','date']]
price_buy_df=price_[price_['type']=='buy']
price_buy_df=price_buy_df.drop('type',1)
price_sell_df=price_[price_['type']=='sell']
price_sell_df=price_sell_df.drop('type',1)

price_buy=price_buy_df['price'].to_numpy()
price_sell=price_sell_df['price'].to_numpy()


# In[468]:


time_=(data1['date']).to_numpy()
time_buy=time_[price_['type']=='buy']
time_sell=time_[price_['type']=='sell']
plt.plot(time_buy, price_buy, '--', label='price buy')
plt.plot(time_sell, price_sell, '--', label='price sell')
plt.title('prices in time')
plt.ylabel('price, BTC/USD')
```

```
plt.xlabel('time')
plt.legend()
plt.show()


# Для оценки ликвидности как один из параметров используется bid-ask spread, который
можно найти в каждую единицу времени, имеющую место в исходном датасете.

# In[469]:


def evaluate_bas (df):
    length=len(df)
    i=0
    res=0
    while i<length:
        j=i
        while (j<length) and (df['type'][j]==df['type'][i]):
            j+=1
        if j >= length:
            j -= 1
        res=df['price'][i]-df['price'][j]
        if (df['type'][i]=='sell'):
            res=0-res
        df['bid-ask spread'][i]=res
        i+=1
    return df


# In[470]:


data=evaluate_bas(data)
```

2.3. Исследуем данные на валидность к обучению и наличие зависимостей

```
# In[476]:


# Округлилим все значения времени до минут
data['date'] = data['date'].values.astype('<M8[m]')


# In[502]:


data_divided=data.groupby('date').mean()
data_divided
```

```
# Посмотрим на зависимость признаков друг от друга:

# In[478]:


corrmat = data_divided.corr()
cols = corrmat.index.tolist()
cols


# In[479]:


cm = np.corrcoef(data_divided[cols].values.T)
plt.figure(figsize=(10, 7))
sns.set(font_scale=1.25)
sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 12},
yticklabels=cols, xticklabels=cols, vmin=-1, center=0,
cmap=sns.color_palette('coolwarm',1000))
plt.show()


# In[480]:


sns.pairplot(data_divided)
plt.show()


# Некоторая корреляция признаков прослеживается, но она нормальна, поэтому нейросеть
должна показать независимые результаты.

# In[504]:


data_divided['time']=data_divided.index
data_divided1=data_divided.iloc[:100]
time_=(data_divided1['time']).to_numpy()


# In[505]:


amount_=data_divided1['amount'].to_numpy()
plt.plot(time_, amount_, '-', label='amount')
plt.title('amount')
plt.ylabel('amount')
plt.xlabel('time')
plt.legend()
```

```
plt.show()


# In[506]:


price_=data_divided1['price'].to_numpy()
plt.plot(time_, price_, '-', label='price')
plt.title('price')
plt.ylabel('price')
plt.xlabel('time')
plt.legend()
plt.show()


# In[507]:


bas_=data_divided1['bid-ask spread'].to_numpy()
plt.plot(time_, bas_, '-', label='bid-ask spread')
plt.title('bid-ask spread')
plt.ylabel('bid-ask spread')
plt.xlabel('time')
plt.legend()
plt.show()


# In[508]:


tr_time_=data_divided1['transaction time'].to_numpy()
plt.plot(time_, tr_time_, '-', label='transaction time')
plt.title('transaction time')
plt.ylabel('transaction time')
plt.xlabel('time')
plt.legend()
plt.show()


# Судя по графикам, наиболее четкая зависимость прослеживается у параметров объёма и
спреда, поэтому, скорее всего, именно они дадут более чёткие прогнозы.
```

2.4. Готовим тензоры для обучения и тестирования модели

```
# # Preprocessing the data

# In[509]:


import torch
```

```
# In[510]:


device = torch.device('cuda:0') if torch.cuda.is_available else torch.device('cpu')
device


# In[511]:


import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim


# In[513]:


length_train=int(2*len(data_divided)/3)
data_train=data_divided.iloc[:length_train]
data_test=data_divided.iloc[length_train+1:]
print(len(data_train),len(data_test),len(data_divided))


# In[514]:


price=torch.from_numpy(np.array(data_train['price']))
amount=torch.from_numpy(np.array(data_train['amount']))
bas=torch.from_numpy(np.array(data_train['bid-ask spread']))
tr_time=torch.from_numpy(np.array(data_train['transaction time']))


# In[515]:


price_test=torch.from_numpy(np.array(data_test['price']))
amount_test=torch.from_numpy(np.array(data_test['amount']))
bas_test=torch.from_numpy(np.array(data_test['bid-ask spread']))
tr_time_test=torch.from_numpy(np.array(data_test['transaction time']))


# In[516]:


# Нормализуем тензоры
from sklearn.preprocessing import MinMaxScaler
from torch.autograd import Variable
```

```
scaler=MinMaxScaler(feature_range=(-1, 1))
price=scaler.fit_transform(price.reshape(-1, 1))
amount=scaler.fit_transform(amount.reshape(-1, 1))
bas=scaler.fit_transform(bas.reshape(-1, 1))
tr_time=scaler.fit_transform(tr_time.reshape(-1, 1))


# In[517]:


price_test = scaler.fit_transform(price_test.reshape(-1, 1))
amount_test=scaler.fit_transform(amount_test.reshape(-1, 1))
bas_test=scaler.fit_transform(bas_test.reshape(-1, 1))
tr_time_test=scaler.fit_transform(tr_time_test.reshape(-1, 1))


# In[518]:


def divide_tensor(tensor, step):
    res_train=[]
    res_labels=[]
    length=len(tensor)
    for i in range(length-step):
        part_train=tensor[i:i+step]
        part_check=tensor[i+step:i+step+1]
        res_train.append(part_train)
        res_labels.append(part_check)
    return res_train,res_labels


# In[519]:


price_train,price_train_labels=divide_tensor(price,60)
price_train=Variable(torch.from_numpy(np.array(price_train)).float())
price_train_labels=Variable(torch.from_numpy(np.array(price_train_labels)).float())

amount_train,amount_train_labels=divide_tensor(amount,60)
amount_train=Variable(torch.from_numpy(np.array(amount_train)).float())
amount_train_labels=Variable(torch.from_numpy(np.array(amount_train_labels)).float())

bas_train,bas_train_labels=divide_tensor(bas,60)
bas_train=Variable(torch.from_numpy(np.array(bas_train)).float())
bas_train_labels=Variable(torch.from_numpy(np.array(bas_train_labels)).float())

tr_time_train,tr_time_train_labels=divide_tensor(tr_time,60)
tr_time_train=Variable(torch.from_numpy(np.array(tr_time_train)).float())
tr_time_train_labels=Variable(torch.from_numpy(np.array(tr_time_train_labels)).float())
```

```
# In[520]:


price_test,price_test_labels=divide_tensor(price_test,60)
price_test=Variable(torch.from_numpy(np.array(price_test)).float())
price_test_labels=Variable(torch.from_numpy(np.array(price_test_labels)).float())

amount_test,amount_test_labels=divide_tensor(amount_test,60)
amount_test=Variable(torch.from_numpy(np.array(amount_test)).float())
amount_test_labels=Variable(torch.from_numpy(np.array(amount_test_labels)).float())

bas_test,bas_test_labels=divide_tensor(bas_test,60)
bas_test=Variable(torch.from_numpy(np.array(bas_test)).float())
bas_test_labels=Variable(torch.from_numpy(np.array(bas_test_labels)).float())

tr_time_test,tr_time_test_labels=divide_tensor(tr_time_test,60)
tr_time_test=Variable(torch.from_numpy(np.array(tr_time_test)).float())
tr_time_test_labels=Variable(torch.from_numpy(np.array(tr_time_test_labels)).float())
```

3. Готовим данные для прогнозирования аналогично пункту 2

```
# # Preprocessing the data for forecasting

# In[533]:


data1 = pd.read_csv('BTCUSD_Bitstamp_Q1_2021_prints.csv')
data1


# In[534]:


data1=data1.drop('trans_id',1)
data1=data1.drop('dollar_amount',1)
data1=data1.drop('unix',1)
data1=data1.drop('symbol',1)
# Откорректируем формат данных столбца с датами
from datetime import datetime
data1['date']=data1['date'].apply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S"))
# Сортируем данные от более старых к более новым
data1=data1.sort_values(by='date')
data1.index=range(0,len(data1))
data1['bid-ask spread']=np.nan
first=data1['date'].to_numpy()
second=np.roll(data1['date'],1)
data1['transaction time']=(first-second)
data1=data1.iloc[1:]
data1['transaction time']=data1['transaction time'].apply(lambda x: x.total_seconds())
data1
```

```
# In[537]:


data1=data1.iloc[-300000:]
data1.index=range(0,len(data1))


# In[538]:


data1=evaluate_bas(data1)
data1['date']=data1['date'].values.astype('<M8[m]')
data1_divided=data1.groupby('date').mean()
data1_divided


# In[539]:


data1_divided=data1_divided.iloc[:int(int(14213/60)*60)]


# In[540]:


price_forecast=torch.from_numpy(np.array(data1_divided['price']))
amount_forecast=torch.from_numpy(np.array(data1_divided['amount']))
bas_forecast=torch.from_numpy(np.array(data1_divided['bid-ask spread']))
tr_time_forecast=torch.from_numpy(np.array(data1_divided['transaction time']))


# In[541]:


scaler=MinMaxScaler(feature_range=(-1, 1))
price_forecast=scaler.fit_transform(price_forecast.reshape(-1, 1)).reshape(236,60,1)
amount_forecast=scaler.fit_transform(amount_forecast.reshape(-1, 1)).reshape(236,60,1)
bas_forecast=scaler.fit_transform(bas_forecast.reshape(-1, 1)).reshape(236,60,1)
tr_time_forecast=scaler.fit_transform(tr_time_forecast.reshape(-1, 1)).reshape(236,60,1)


# In[542]:


price_forecast=Variable(torch.from_numpy(np.array(price_forecast)).float())
amount_forecast=Variable(torch.from_numpy(np.array(amount_forecast)).float())
bas_forecast=Variable(torch.from_numpy(np.array(bas_forecast)).float())
tr_time_forecast=Variable(torch.from_numpy(np.array(tr_time_forecast)).float())
```

4. Проводим три этапа работы нейросети для каждого из параметров
   4.1. Класс используемой модели

```
# # Training

# In[521]:
```

```
seq_length=60
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()
        self.num_classes=num_classes
        self.num_layers=num_layers
        self.input_size=input_size
        self.hidden_size=hidden_size
        self.seq_length=seq_length
        self.lstm=nn.LSTM(input_size=input_size, hidden_size=hidden_size, num_lay-
ers=num_layers, batch_first=True)
        self.fc=nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0=Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        c_0=Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        # Propagate input through LSTM
        ula, (h_out, _)=self.lstm(x, (h_0, c_0))
        h_out=h_out.view(-1, self.hidden_size)
        out=self.fc(h_out)
        return out
```

4.2. Блок для параметра «цена»

```
# ## 1. Price

# Тренируем

# In[604]:


num_epochs=50
learning_rate=0.01
input_size=1
hidden_size=2
num_layers=1
num_classes=1
history=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

# Обучаем модель
for epoch in range(num_epochs):
    outputs=lstm(price_train)
    optimizer.zero_grad()
```

```
    # Вычисляем потери
    loss=criterion(outputs, price_train_labels)
    loss.backward()
    history.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# Тестируем

# In[605]:


history_test=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

# Тестируем
for epoch in range(num_epochs):
    outputs=lstm(price_test)
    optimizer.zero_grad()

    loss=criterion(outputs, price_test_labels)
    loss.backward()
    history_test.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# In[607]:


price_test_labels=price_test_labels.reshape(-1,1)
outputs=outputs.detach().numpy()
mae=mean_absolute_error(price_test_labels,outputs)
rmse=mean_squared_error(price_test_labels,outputs) ** 0.5
r2=r2_score(price_test_labels,outputs)
print("MAE: %1.5f, RMSE: %1.5f, R2: %1.5f" % (mae,rmse,r2))


# In[608]:


plt.figure(figsize=(10, 7))
plt.plot(history,label='Training loss')
plt.plot(history_test,label='Validation loss')
plt.title('Price loss by batch iterations')
```

```
plt.ylabel('Entropy Loss')
plt.xlabel('batches')
plt.show()


# Делаем прогноз

# In[610]:


with torch.no_grad():
    output=lstm(price_forecast)[-1]
scaler.inverse_transform(output.reshape(-1,1))


# In[611]:


scaler.inverse_transform(price_forecast[-1][-1].reshape(-1,1))
```

4.3. Блок для параметра «объём»

```
# ## 2. Amount

# Тренируем

# In[598]:


history=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    outputs=lstm(amount_train)
    optimizer.zero_grad()

    loss=criterion(outputs, amount_train_labels)
    loss.backward()
    history.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# Тестируем

# In[599]:
```

```
history_test=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    outputs=lstm(amount_test)
    optimizer.zero_grad()

    loss=criterion(outputs, amount_test_labels)
    loss.backward()
    history_test.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# In[600]:


amount_test_labels=amount_test_labels.reshape(-1,1)
outputs=outputs.detach().numpy()
mae=mean_absolute_error(amount_test_labels,outputs)
rmse=mean_squared_error(amount_test_labels,outputs) ** 0.5
r2=r2_score(amount_test_labels,outputs)
print("MAE: %1.5f, RMSE: %1.5f, R2: %1.5f" % (mae,rmse,r2))


# In[601]:


plt.figure(figsize=(10, 7))
plt.plot(history,label='Training loss')
plt.plot(history_test,label='Validation loss')
plt.title('Amount loss by batch iterations')
plt.ylabel('Entropy Loss')
plt.xlabel('batches')
plt.show()


# Делаем прогноз

# In[602]:


with torch.no_grad():
    output=lstm(amount_forecast)[-1]
scaler.inverse_transform(output.reshape(-1,1))
```

```
# In[603]:


scaler.inverse_transform(amount_forecast[-1][-1].reshape(-1,1))
```

4.4. Блок для параметра «бид-аск спред»

```
# ## 3. Bid-ask spread

# Тренируем

# In[580]:


history=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    outputs=lstm(bas_train)
    optimizer.zero_grad()

    loss=criterion(outputs, bas_train_labels)
    loss.backward()
    history.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# Тестируем

# In[583]:


history_test=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    outputs=lstm(bas_test)
    optimizer.zero_grad()

    loss=criterion(outputs, bas_test_labels)
    loss.backward()
    history_test.append(loss.item())
    optimizer.step()
```

```
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# In[587]:


bas_test_labels=bas_test_labels.reshape(-1,1)
outputs=outputs.detach().numpy()
mae=mean_absolute_error(bas_test_labels,outputs)
rmse=mean_squared_error(bas_test_labels,outputs) ** 0.5
r2=r2_score(bas_test_labels,outputs)
print("MAE: %1.5f, RMSE: %1.5f, R2: %1.5f" % (mae,rmse,r2))


# In[588]:


plt.figure(figsize=(10, 7))
plt.plot(history,label='Training loss')
plt.plot(history_test,label='Validation loss')
plt.title('Bid-ask spread loss by batch iterations')
plt.ylabel('Entropy Loss')
plt.xlabel('batches')
plt.show()


# Делаем прогноз

# In[589]:


with torch.no_grad():
    output=lstm(bas_forecast)[-1]
scaler.inverse_transform(output.reshape(-1,1))


# In[590]:


scaler.inverse_transform(bas_forecast[-1][-1].reshape(-1,1))
```

4.5. Блок для параметра «время транзакции»

```
# ## 4. Transaction time

# Тренируем

# In[591]:
```

```
history=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    outputs=lstm(tr_time_train)
    optimizer.zero_grad()

    loss=criterion(outputs, tr_time_train_labels)
    loss.backward()
    history.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# Тестируем

# In[592]:


history_test=[]
lstm=LSTM(num_classes, input_size, hidden_size, num_layers)
criterion=torch.nn.MSELoss()
optimizer=torch.optim.Adam(lstm.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    outputs=lstm(tr_time_test)
    optimizer.zero_grad()

    loss=criterion(outputs, tr_time_test_labels)
    loss.backward()
    history_test.append(loss.item())
    optimizer.step()
    if epoch%10==9:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))


# In[594]:


tr_time_test_labels=tr_time_test_labels.reshape(-1,1)
outputs=outputs.detach().numpy()
mae=mean_absolute_error(tr_time_test_labels,outputs)
rmse=mean_squared_error(tr_time_test_labels,outputs) ** 0.5
r2=r2_score(tr_time_test_labels,outputs)
print("MAE: %1.5f, RMSE: %1.5f, R2: %1.5f" % (mae,rmse,r2))
```

```
# In[595]:


plt.figure(figsize=(10, 7))
plt.plot(history,label='Training loss')
plt.plot(history_test,label='Validation loss')
plt.title('Transaction time loss by batch iterations')
plt.ylabel('Entropy Loss')
plt.xlabel('batches')
plt.show()


# Делаем прогноз

# In[596]:


with torch.no_grad():
    output=lstm(tr_time_forecast)[-1]
scaler.inverse_transform(output.reshape(-1,1))


# In[597]:


scaler.inverse_transform(tr_time_forecast[-1][-1].reshape(-1,1))
```